What Is Claimed Is:

1. A method for executing a commit instruction to facilitate transactional execution on a processor, comprising:

encountering the commit instruction during execution of a program, wherein the commit instruction marks the end of a block of instructions to be executed transactionally; and

upon encountering the commit instruction, successfully completing transactional execution of the block of instructions preceding the commit instruction;

wherein changes made during the transactional execution are not committed to the architectural state of the processor until the transactional execution successfully completes.

2. The method of claim 1, wherein successfully completing the transactional execution involves:

atomically committing changes made during the transactional execution; and

resuming normal non-transactional execution.

3. The method of claim 2, wherein atomically committing changes made during the transactional execution involves:

treating store-marked cache lines as locked, thereby causing other processes to wait to access the store-marked cache lines;

clearing load marks from cache lines;

committing store buffer entries generated during transactional execution to memory, wherein committing each store buffer entry involves unmarking, and thereby unlocking, a corresponding store-marked cache line; and

committing register file changes made during transactional execution.

4. The method of claim 1, wherein if an interfering data access from another process is encountered during the transactional execution and prior to encountering the commit instruction, the method further comprises:

discarding changes made during the transactional execution; and

attempting to re-execute the block of instructions.

5. The method of claim 1, wherein for a variation of the commit instruction, successfully completing the transactional execution involves:

atomically committing changes made during the transactional execution; and

commencing transactional execution of the block of instructions following the commit instruction.

6. The method of claim 1, wherein potentially interfering data accesses from other processes are allowed to proceed during the transactional execution of the block of instructions.

7. The method of claim 1, wherein the block of instructions to be executed transactionally comprises a critical section.

8. The method of claim 1, wherein the commit instruction is a native machine code instruction of the processor.

9. The method of claim 1, wherein the commit instruction is defined in a platform-independent programming language.

10. A computer system that supports a commit instruction to facilitate transactional execution, wherein the commit instruction marks the end of a block of instructions to be executed transactionally, comprising:

a processor; and

an execution mechanism within the processor;

wherein upon encountering the commit instruction, the execution mechanism is configured to successfully complete transactional execution of the block of instructions preceding the commit instruction;

wherein changes made during the transactional execution are not committed to the architectural state of the processor until the transactional execution successfully completes.

11. The computer system of claim 10, wherein while successfully completing transactional execution, the execution mechanism is configured to:

atomically commit changes made during the transactional execution; and to

resume normal non-transactional execution.

12. The computer system of claim 11, wherein while atomically committing changes made during the transactional execution, the execution mechanism is configured to:

treat store-marked cache lines as locked, thereby causing other processes to wait to access the store-marked cache lines;

clear load marks from cache lines;

commit store buffer entries generated during transactional execution to memory, wherein committing each store buffer entry involves unmarking, and thereby unlocking, a corresponding store-marked cache line; and to

commit register file changes made during transactional execution.

13. The computer system of claim 10, wherein if an interfering data access from another process is encountered during the transactional execution and prior to encountering the commit instruction, the execution mechanism is configured to:

discard changes made during the transactional execution; and to

attempt to re-execute the block of instructions.

14. The computer system of claim 10, wherein if a variation of the commit instruction is encountered, the execution mechanism is configured to:

atomically commit changes made during the transactional execution; and to

commence transactional execution of the block of instructions following the commit instruction.

15. The computer system of claim 10, wherein the computer system is configured to allow potentially interfering data accesses from other processes to proceed during the transactional execution of the block of instructions.

16. The computer system of claim 10, wherein the block of instructions to be executed transactionally comprises a critical section.

17. The computer system of claim 10, wherein the commit instruction is a native machine code instruction of the processor.

18. The computer system of claim 10, wherein the commit instruction is defined in a platform-independent programming language.

19. A computing means that supports a commit instruction to facilitate transactional execution, wherein the commit instruction marks the end of a block of instructions to be executed transactionally, comprising:

a processing means; and

an execution means within the processing means;

wherein upon encountering the commit instruction, the execution means is configured to successfully complete transactional execution of the block of instructions preceding the commit instruction;

wherein changes made during the transactional execution are not committed to the architectural state of the processor until the transactional execution successfully completes.

20. The computing means of claim 19, wherein while successfully completing transactional execution, the execution means is configured to:

atomically commit changes made during the transactional execution; and to

resume normal non-transactional execution.